Algoritmos de Ordenação

Allan Robson

DCA

Estruturas de Dados

Algoritmos de Ordenação

Introdução

- **Ordenação** é a tarefa de colocar um conjunto de dados em uma determinada ordem;
- Algoritmo de ordenação:
  - É o algoritmo que coloca os elementos de uma dada sequência em uma certa ordem.
- Ao se ordenar uma sequencia, seus dados podem ser acessados de forma mais eficiente;
- Exemplo:
  - [5 2 1 3 4] → Fora de Ordem
  - [1 2 3 4 5] → Ordenado

- Definição Formal:
  - Dado um vetor de n elemento:

$$v = (v_1, v_2, \dots, v_n)$$

  - Encontrar uma permutação v^* de v tal que:

$$v_i^* \leq v_j^*, \forall i < j \text{ com } i \text{ e } j \in \{1, \dots, n\}$$

- Tipos de Ordenação:
  - Numérica
    - 1, 2, 3, 4
  - Lexicográfica (ordem alfabética)
    - Ana, André, Bianca, Ricardo

- Independente do tipo ela pode ser:
  - Crescente:
    - 1, 2, 3, 4, 5
    - Ana, André, Bianca, Ricardo
  - Decrescente
    - 5, 4, 3, 2, 1
    - Ricardo, Bianca, André, Ana

- Existem vários algoritmos de ordenação, que serão abordados a seguir:
  - *Bubble sort;*
  - *Insertion sort;*
  - *Merge sort;*
  - *Quick sort*;
  - ...

Bubble Sort

Bubble Sort

Introdução

- *Bubble Sort* ou Ordenação por Bolha é um dos algoritmos de ordenação mais simples;
- **Ele compara pares de elementos adjacentes e os troca de lugar se estiverem na ordem errada;**
- Esse processo se repete até que mais nenhuma troca seja necessária (elementos já ordenados)

Bubble Sort

Complexidade

- Complexidade do algoritmo:
  - **Pior caso: $O(n^2)$**

- Esse algoritmo não é recomendado para grandes conjuntos de dados.

- Algoritmos deste tipo somente são úteis para resolver problemas de tamanho relativamente pequenos.

Bubble Sort

Algoritmo

BUBBLESORT($v[\,], n$)

var houveTroca = true

enquanto (*houveTroca*) for verdade faça:

houveTroca = false

para i de 0 até $(n-1)$ faça:

se $(v[i] > v[i+1])$ então:

troque $v[i]$ e $v[i+1]$ de posição

houveTroca = true

fim se

fim para

fim enquanto

fim BUBBLESORT

Bubble Sort

Exemplo

- Ordenar o vetor:

23	4	67	-8	90	54	21

- 1ª Iteração:

$i = 0$	23	4	67	-8	90	54	21	**Trocar**
$i = 1$	4	23	67	-8	90	54	21	**Ok**
$i = 2$	4	23	67	-8	90	54	21	**Trocar**
$i = 3$	4	23	-8	67	90	54	21	**Ok**
$i = 4$	4	23	-8	67	90	54	21	**Trocar**
$i = 5$	4	23	-8	67	54	90	21	**Trocar**
Final	4	23	-8	67	54	21	90	

Bubble Sort

Exemplo

- Ordenar o vetor:

23	4	67	-8	90	54	21

- 1ª Iteração:

$i = 0$	23	4	67	-8	90	54	21	**Trocar**
$i = 1$	4	23	67	-8	90	54	21	**Ok**
$i = 2$	4	23	67	-8	90	54	21	**Trocar**
$i = 3$	4	23	-8	67	90	54	21	**Ok**
$i = 4$	4	23	-8	67	90	54	21	**Trocar**
$i = 5$	4	23	-8	67	54	90	21	**Trocar**
Final	4	23	-8	67	54	21	90	

Ao fim da primeira iteração, o maior elemento estará no fim do vetor

Bubble Sort

Exemplo

2ª Iteração

$i = 0$	4	23	-8	67	54	21	90	Ok
$i = 1$	4	23	-8	67	54	21	90	Trocar
$i = 2$	4	-8	23	67	54	21	90	Ok
$i = 3$	4	-8	23	67	54	21	90	Trocar
$i = 4$	4	-8	23	54	67	21	90	Trocar
$i = 5$	4	-8	23	54	21	67	90	Ok
Final	4	-8	23	54	21	67	90	

3ª Iteração

$i = 0$	4	-8	23	54	21	67	90	Trocar
$i = 1$	-8	4	23	54	21	67	90	Ok
$i = 2$	-8	4	23	54	21	67	90	Ok
$i = 3$	-8	4	23	54	21	67	90	Trocar
$i = 4$	-8	4	23	21	54	67	90	Ok
$i = 5$	-8	4	23	21	54	67	90	Ok
Final	-8	4	23	21	54	67	90	

4ª Iteração

$i = 0$	-8	4	23	21	54	67	90	Ok
$i = 1$	-8	4	23	21	54	67	90	Ok
$i = 2$	-8	4	23	21	54	67	90	Trocar
$i = 3$	-8	4	21	23	54	67	90	Ok
$i = 4$	-8	4	21	23	54	67	90	Ok
$i = 5$	-8	4	21	23	54	67	90	Ok
Final	-8	4	21	23	54	67	90	

5ª Iteração

$i = 0$	-8	4	21	23	54	67	90	Ok
$i = 1$	-8	4	21	23	54	67	90	Ok
$i = 2$	-8	4	21	23	54	67	90	Ok
$i = 3$	-8	4	21	23	54	67	90	Ok
$i = 4$	-8	4	21	23	54	67	90	Ok
$i = 5$	-8	4	21	23	54	67	90	Ok
Final	-8	4	21	23	54	67	90	

Insertion Sort

- Eficiente quando aplicado a um número pequeno de elementos;

- Percorre um vetor da esquerda para a direita e, conforme avança, os elementos mais à esquerda ficam ordenados;

- No geral, o funcionamento é semelhante à ordenação de cartas de um jogo de baralho;
  - Pega-se uma carta de cada vez e a coloca em seu devido lugar, sempre deixando as cartas da mão em ordem.

Complexidade

- Quando o vetor já está ordenado, ele faz o menor número de comparações.

- Pior caso quando o vetor está ordenado de forma decrescente.

- Complexidade do pior caso:
  - $O(n^2)$

INSERTION SORT

Algoritmo

INSERTIONSORT($v[\,], n$)

var aux, j

para i de 1 até n faça:

$aux = v[i]$

$j = i$

enquanto ($j > 0$) & ($aux < v[j-1]$)

$v[j] = v[j-1]$

$j = j - 1$

fim enquanto

$v[j] = aux$

fim para

fim BUBBLESORT

Insertion Sort

Exemplo

- Ordenar o vetor:

23	4	67	-8	90	54	21

$i = 1$	23	4	67	-8	90	54	21	Trocar
Final	4	23	67	-8	90	54	21	

$i = 2$	4	23	67	-8	90	54	21	Ok
Final	4	23	67	-8	90	54	21	

$i = 3$	4	23	67	-8	90	54	21	Trocar
Final	-8	4	23	67	90	54	21	

$i = 4$	-8	4	23	67	90	54	21	Ok
Final	-8	4	23	67	90	54	21	

$i = 5$	-8	4	23	67	90	54	21	Troca
Final	-8	4	23	54	67	90	21	

$i = 6$	-8	4	23	54	67	90	21	Troca
Final	-8	4	21	23	54	67	90	

Insertion Sort

Exemplo

- Ordenar o vetor:

23	4	67	-8	90	54	21

$i = 1$	23	4	67	-8	90	54	21	Trocar
Final	4	23	67	-8	90	54	21	

$i = 2$	4	23	67	-8	90	54	21	Ok
Final	4	23	67	-8	90	54	21	

$i = 3$	4	23	67	-8	90	54	21	Trocar
Final	-8	4	23	67	90	54	21	

$i = 4$	-8	4	23	67	90	54	21	Ok
Final	-8	4	23	67	90	54	21	

$i = 5$	-8	4	23	67	90	54	21	Troca
Final	-8	4	23	54	67	90	21	

$i = 6$	-8	4	23	54	67	90	21	Troca
Final	-8	4	21	23	54	67	90	

Vetor Ordenado

-8	4	21	23	54	67	90

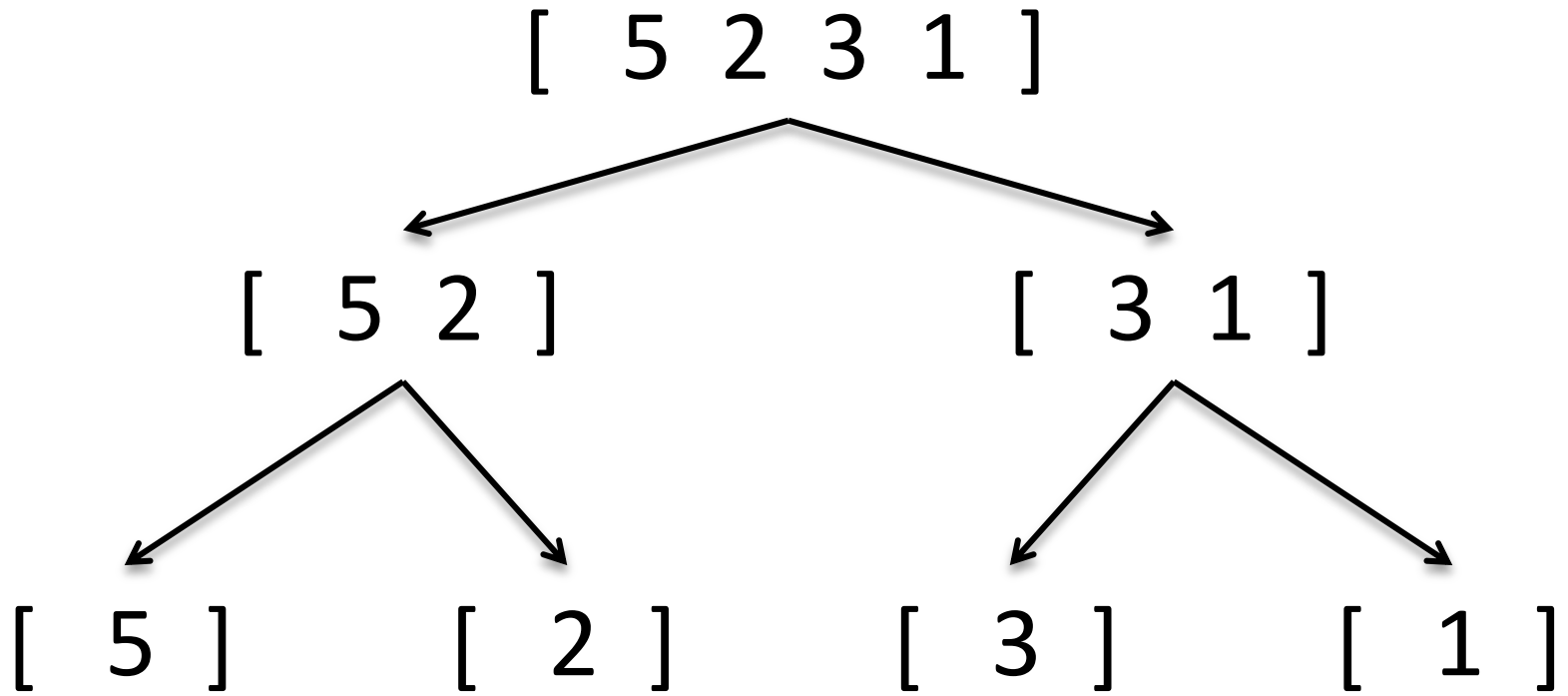
Merge Sort

- **Ideia básica:** Dividir para conquistar:

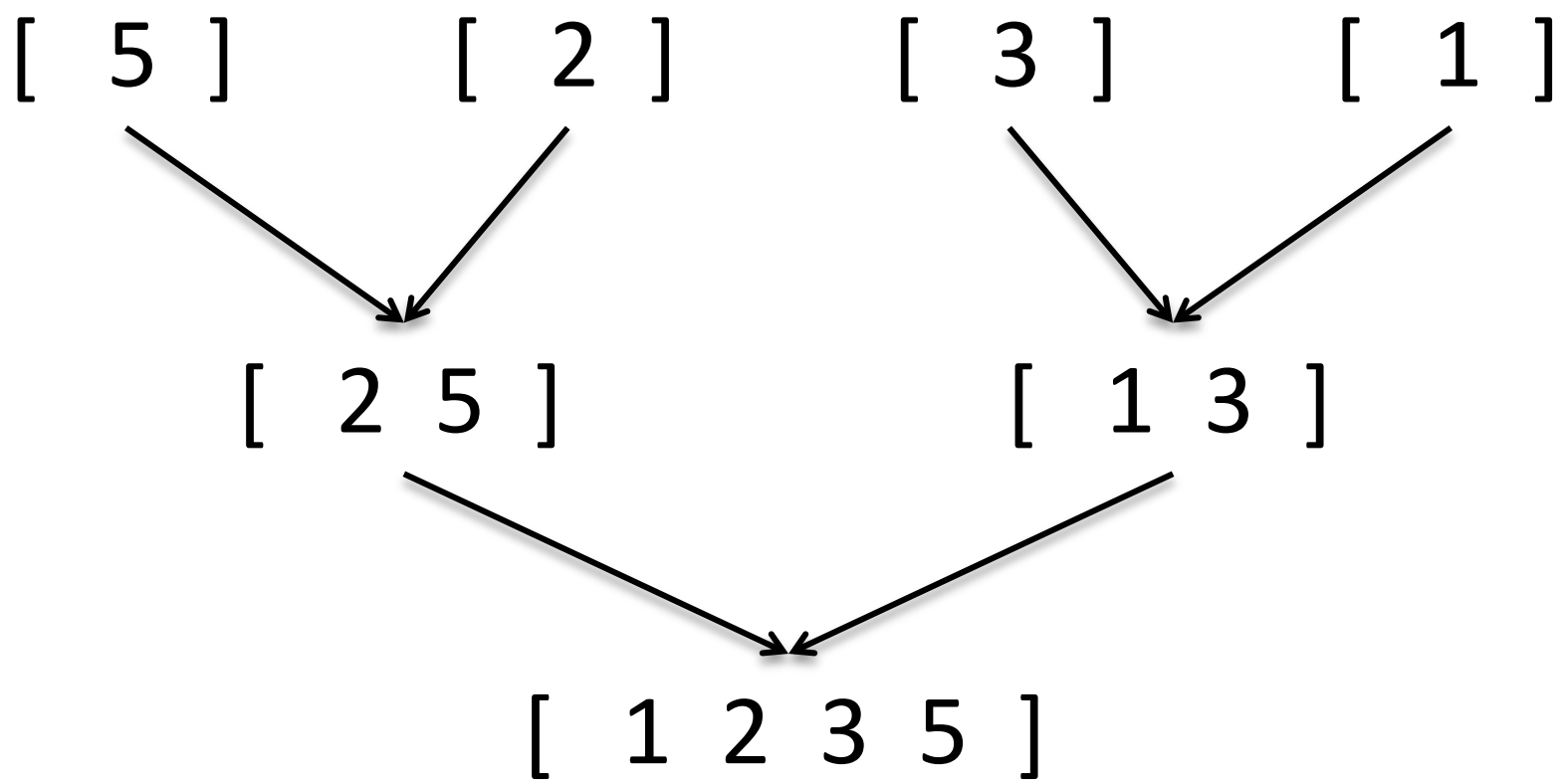
- Divide (**recursivamente**) o conjunto de dados até que cada subconjunto possua 1 elemento.

- **Combina** 2 subconjuntos de forma a obter 1 conjunto maior e ordenado.

- Dividir:



- Combinar:



- **Vantagens**:
  - Facilidade de implementação
  - Recomendado para ordenação de vetores grande
  - Estável: não altera a ordem de dados iguais

- **Desvantagens**
  - Requer o dobro de memória (recursão)
  - Exige uma segunda lista de mesmo tamanho

- **Complexidade**:
  - Pior caso: $\theta(n \log(n))$
  - Caso médio: $\theta(n \log(n))$
  - Melhor caso: $\theta(n \log(n))$

Merge Sort

Algoritmo

$$\boldsymbol{MERGESORT}(v[\,], inicio, fim)$$
$$SE\ (inicio < fim)\ então:$$
$$var\ meio\ = piso\big((inicio + fim)/2\big)$$
$$\boldsymbol{mergesort}(v, inicio, meio)$$
$$\boldsymbol{mergesort}(v, meio + 1, fim)$$
$$\boldsymbol{merge}(v, inicio, meio, fim)$$
$$fim\ SE$$
$$fim\ MERGESORT$$

Merge Sort

Algoritmo

$$\boldsymbol{MERGESORT}(v[\,], inicio, fim)$$

$$SE\ (inicio < fim)\ então:$$

$$var\ meio\ = piso\big((inicio + fim)/2\big)$$

$$\boldsymbol{mergesort}(v, inicio, meio)$$

$$\boldsymbol{mergesort}(v, meio + 1, fim)$$

→ *Dividir os dados*

$$\boldsymbol{merge}(v, inicio, meio, fim)$$

→ *Combina duas metades de forma ordenada*

$$fim\ SE$$

$$fim\ MERGESORT$$

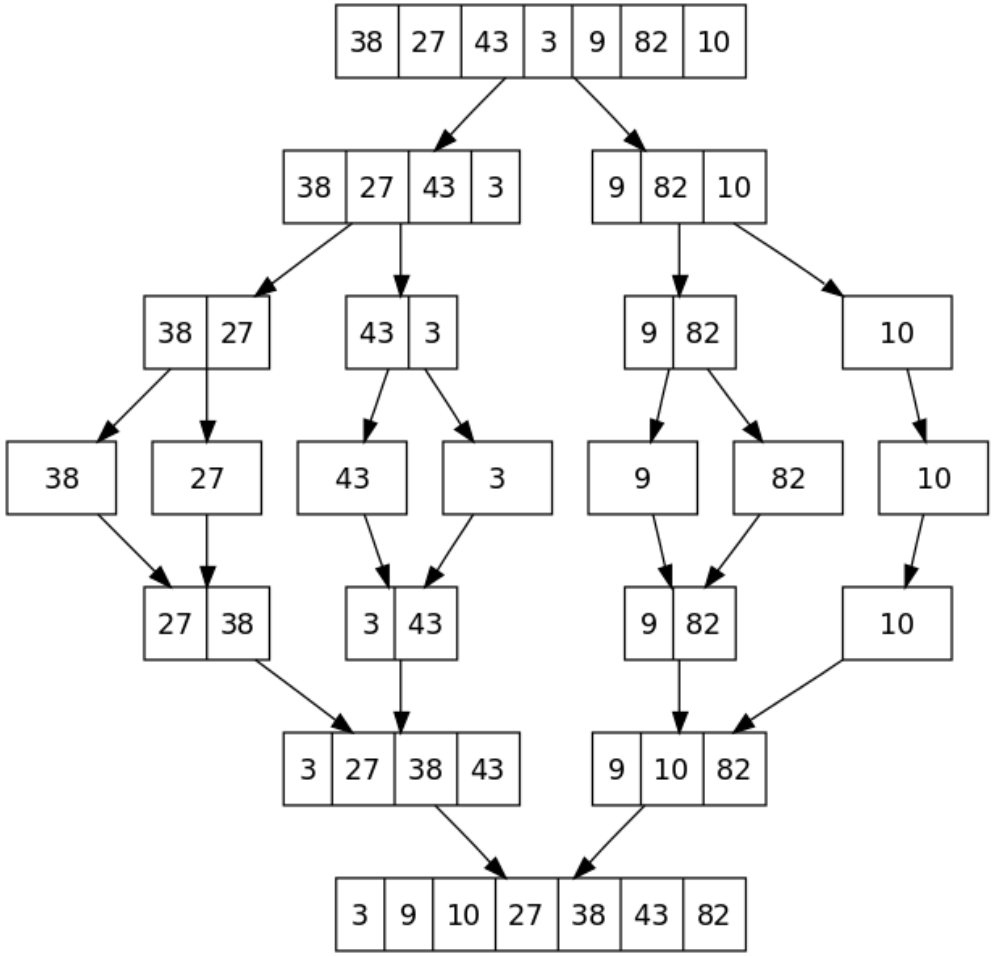
Merge Sort

Algoritmo

```
MERGE(v[ ], inicio, meio, fim)
        var temp[ ]
        var p1 = inicio
        var p2 = meio + 1
        var tamanho = fim − inicio + 1
        PARA (i = 0) até i < tamanho
                SE ((p1 ≤ meio) && (p2 ≤ fim))
                        SE (v[p1] < v[p2])
                                temp[i] = v[p1]
                                p1 + +
                        CASO CONTRÁRIO
                                temp[i] = v[p2]
                                p2 + +
                        fim SE
                CASO CONTRÁRIO
                        SE (p1 ≤ meio)
                                temp[i] = v[p1]
                                p1 + +
                        CASO CONTRÁRIO
                                temp[i] = v[p2]
                                p2 + +
                        fim SE
                fim SE
        fim PARA
        PARA (j = 0 e k = inicio) ate (j < tamanho)
                v[k] = temp[j]
        fim PARA
fim MERGE
```

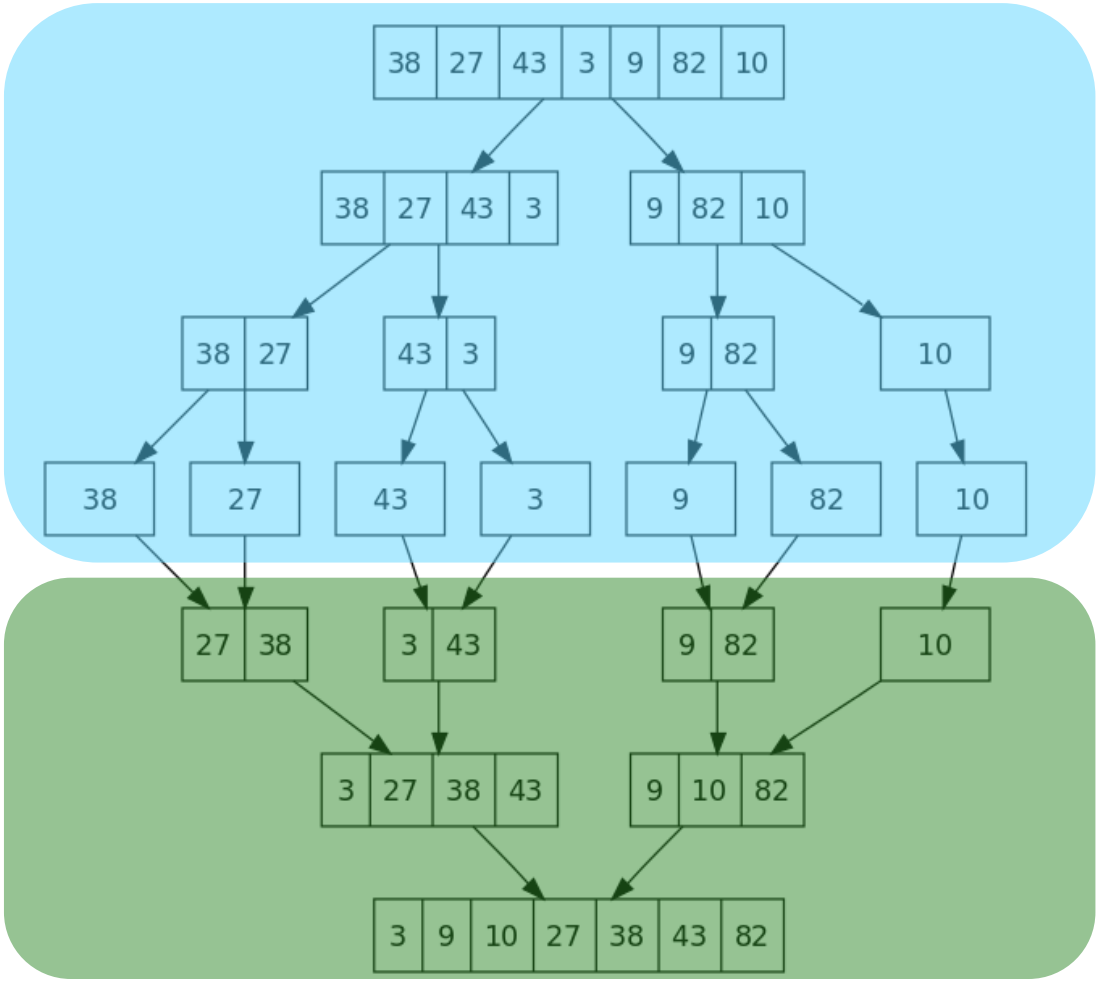
Merge Sort

- **Exemplo**



Merge Sort

Exemplo



Quick Sort

Quick Sort

Introdução

- Também conhecido como Ordenação por Troca de Partições;
- Ideia básica: **dividir e conquistar**
- Um elemento é escolhido como **pivô**;
- **Particionar**:
  - Os dados são rearranjados;
  - Valores menores que o pivô são colocados antes dele e os maiores colocados depois;
- **Recursivamente** ordena as partições.

- **Complexidade:**
  - Pior caso: $\theta(n^2)$ (*raro*)
  - Caso médio: $\theta(n \log(n))$
  - Melhor caso: $\theta(n \log(n))$

- Desvantagem:
  - Como escolher o pivô?

Quick Sort

Algoritmo

$$QUICKSORT(v[\,], inicio, fim)$$
$$\quad se\ (inicio < fim)\ então:$$
$$\quad\quad var\ pivo\ = particiona(v, inicio, fim)$$
$$\quad\quad \boldsymbol{quicksort}(v, inicio, pivo - 1)$$
$$\quad\quad \boldsymbol{quicksort}(v, pivo + 1, fim)$$
$$\quad fim\ se$$
$$fim\ QUICKSORT$$

QUICK SORT

Algoritmo

$PARTICIONA(v[\,], inicio, fim)$
$var\ esq\ = inicio, dir\ = fim$
$var\ pivo\ = v[inicio]$
$enquanto\ (esq < dir)$
$enquanto\ (v[esq] \leq pivo)\ então:$
$esq++$
$fim\ enquanto$
$enquanto\ (v[dir] > pivo)\ então:$
$dir--$
$fim\ enquanto$
$se\ (esq\ <\ dir)\ então:$
troque $v[esq]$ *e* $v[dir]$ *de posição*
$fim\ se$
$fim\ enquanto$
$v[inicio] = v[dir]$
$v[dir] = pivo$
$\boldsymbol{retorna\ dir}$
$fim\ PARTICIONA$

Quick Sort

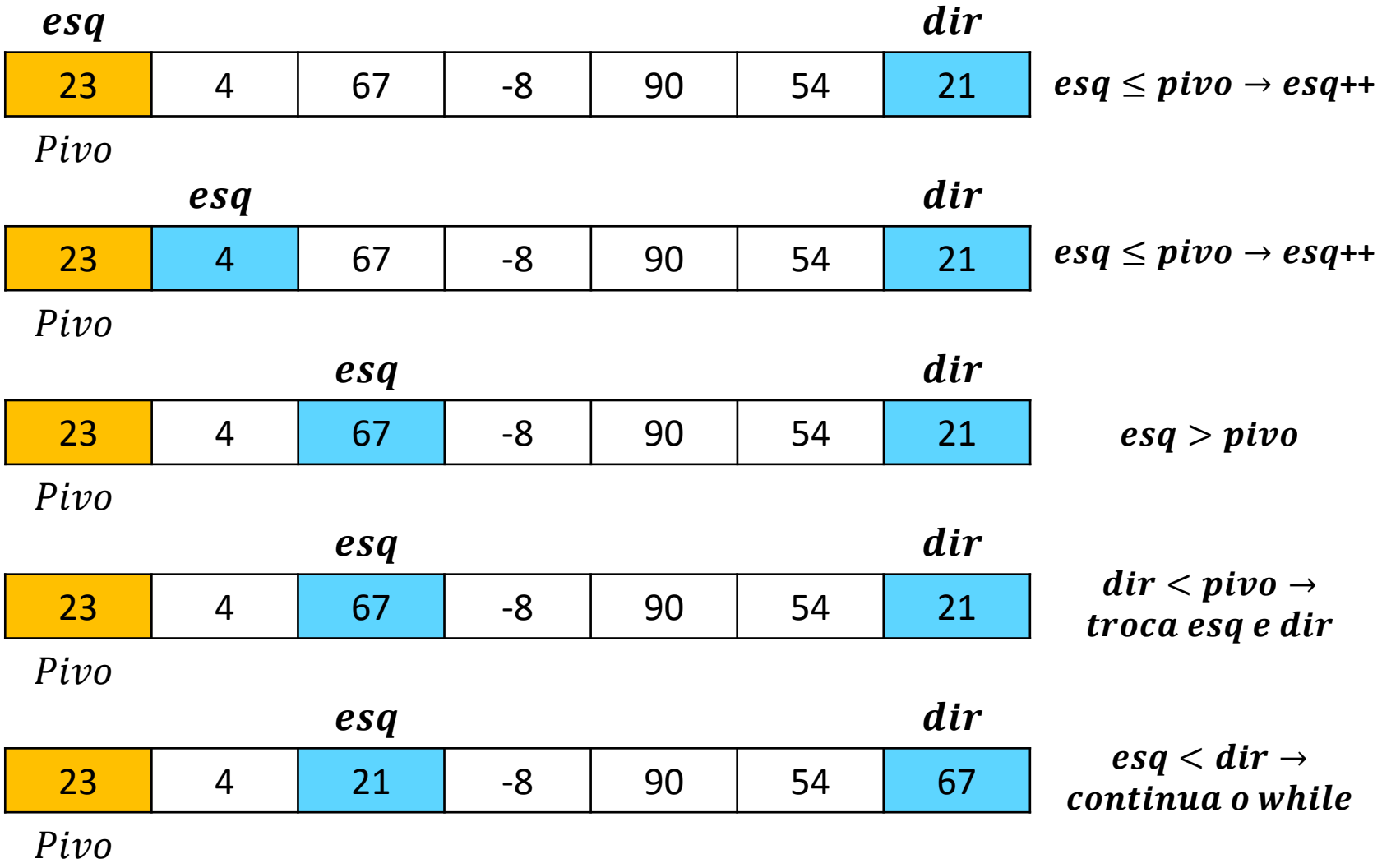
Algoritmo

- Ordenar o vetor:

23	4	67	-8	90	54	21

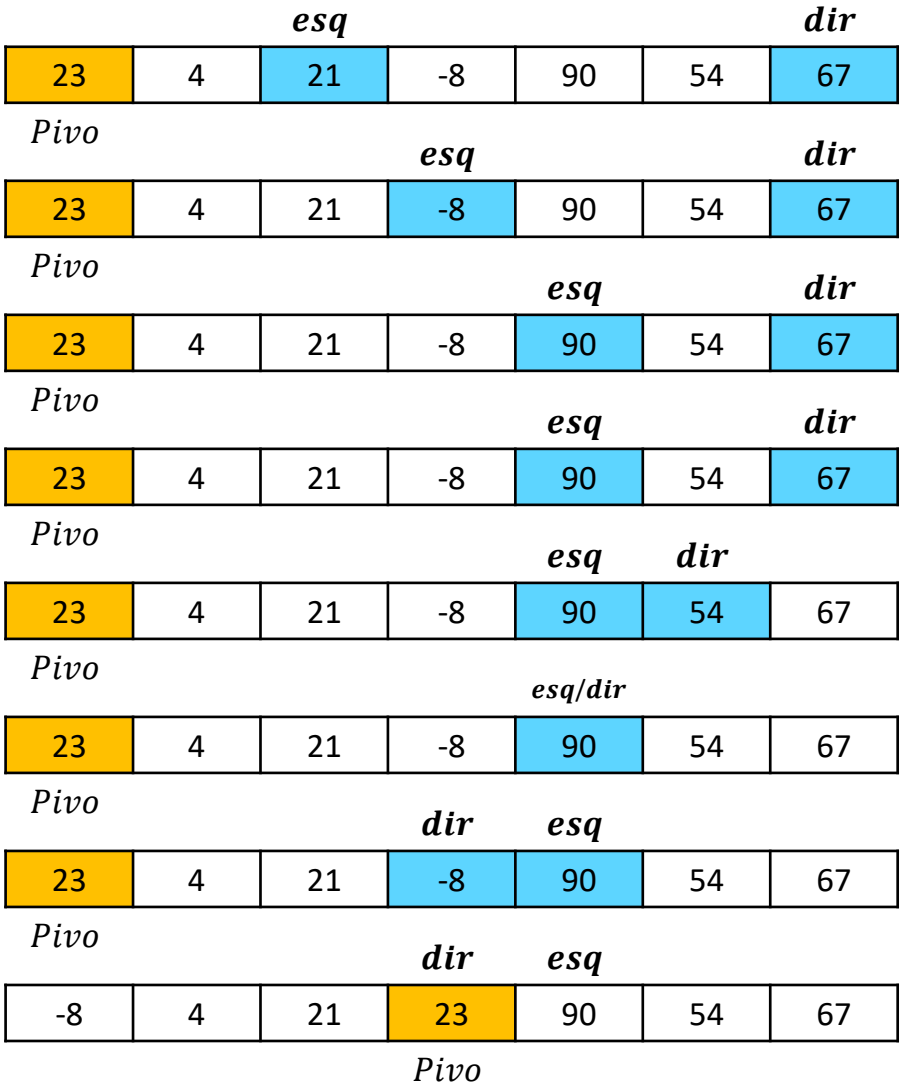
Quick Sort

Algoritmo



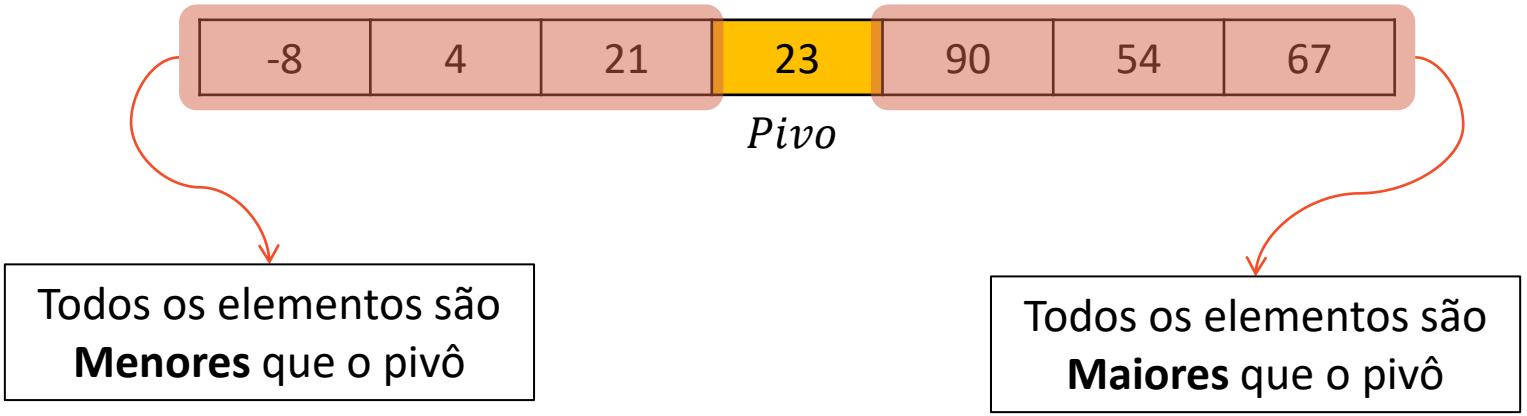
Quick Sort

Algoritmo



Quick Sort

- **Algoritmo**



Quick Sort

- **Algoritmo**

Vetor não ordenado

23	4	67	-8	90	54	21

Quick Sort

Algoritmo

Vetor não ordenado

23	4	67	-8	90	54	21

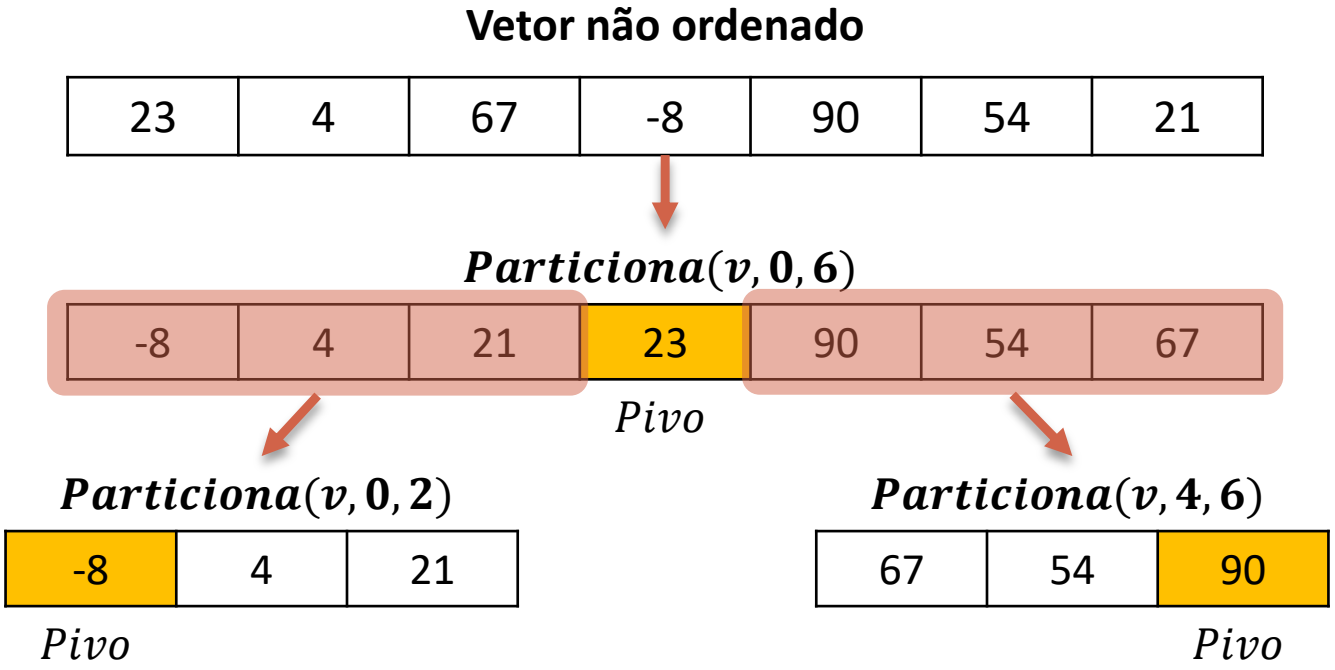
$\boldsymbol{Particiona(v, 0, 6)}$

-8	4	21	23	90	54	67

$Pivo$

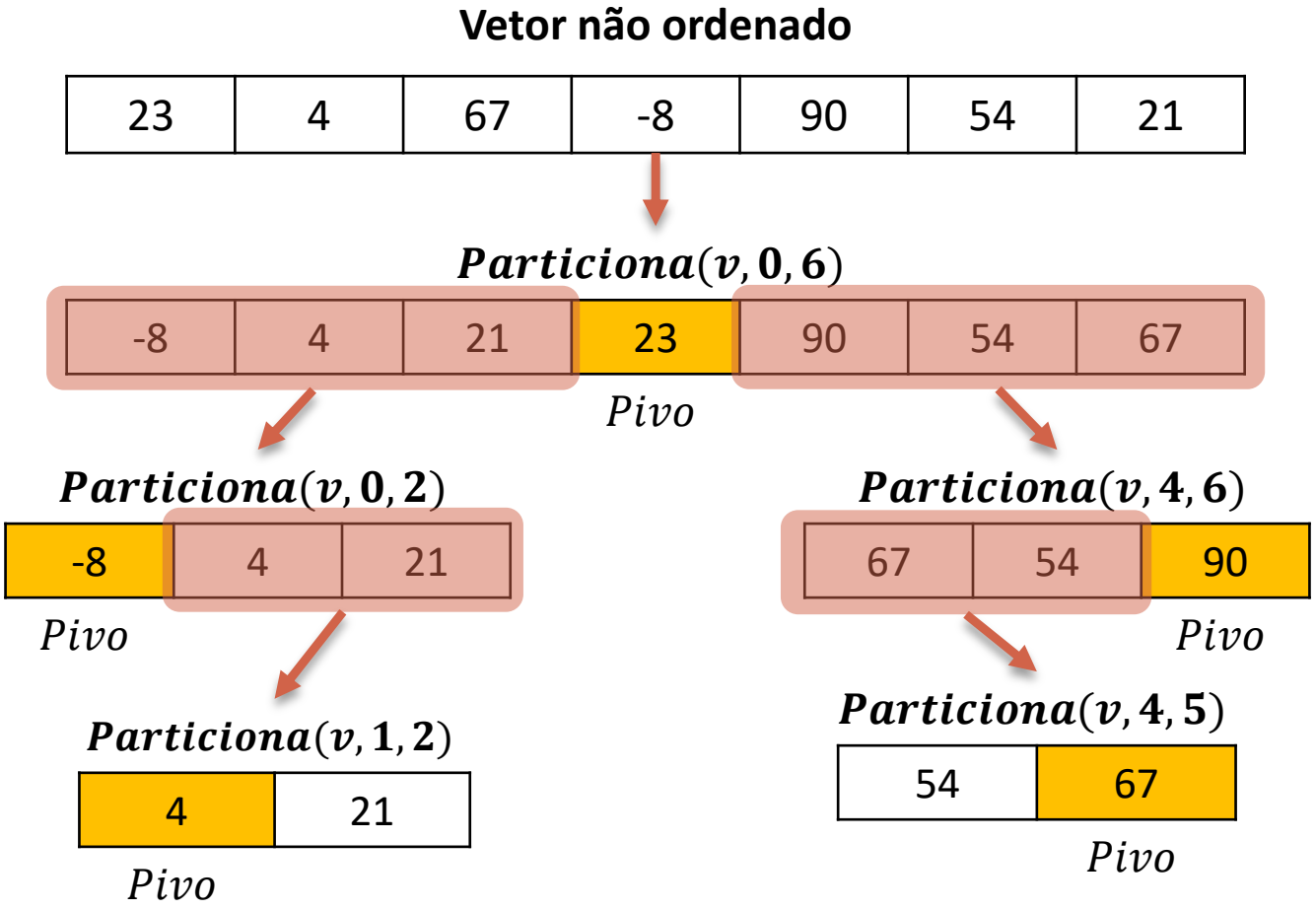
Quick Sort

- **Algoritmo**



Quick Sort

- **Algoritmo**



Quick Sort

- **Algoritmo**